Activity Time Series Classification Using Deep Learning Models

Harry Graves and Christopher Ravenscroft

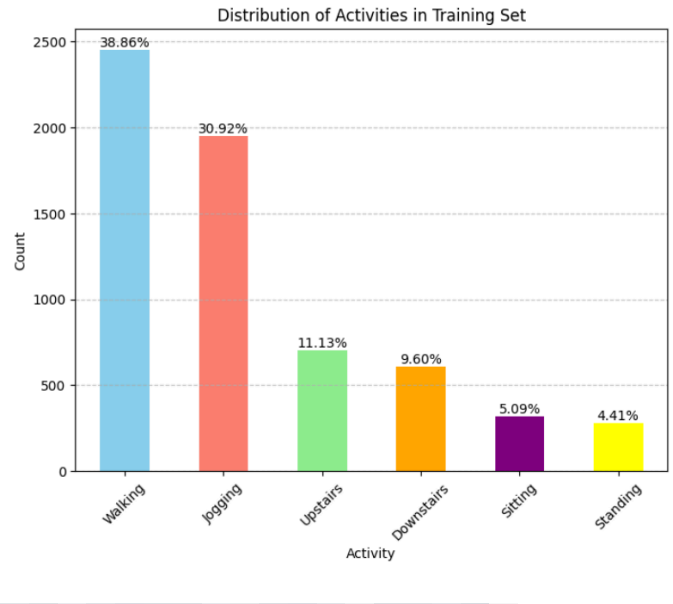
March 2023

Introduction

In the following project, we seek to explore a synthetic replication of the WISDM dataset. The original dataset was produced by Fordham University and recorded 1,098,207 different examples of people in 6 different activities. We however only used a synthetic replication of this data for our project. Our training set contained 629,866 total entries, which map to 6310 instances of activities. The distribution of our training set is shown in Figure 1.

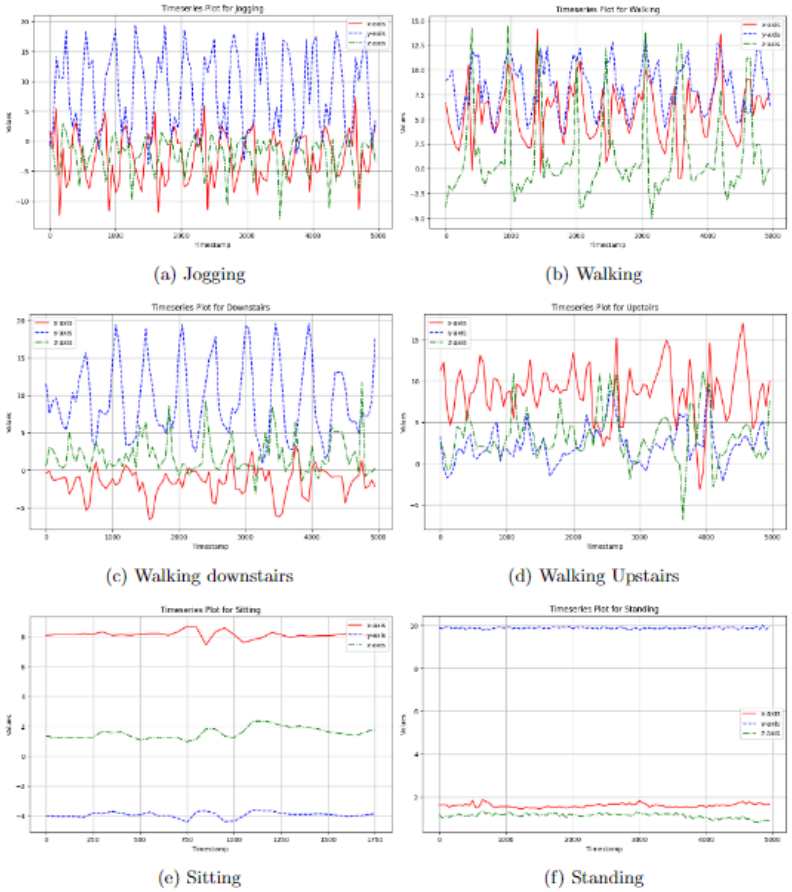
We aim to build multiple machine learning models to identify and predict these activities from acceleration time series data in the z,y, and z direction. To evaluate which model is the best at predicting activity, we will compare the models through the classification report and prediction confusion matrices, more specifically the f1 scores, and weighted accuracy due to the imbalanced nature of the synthetic data.

Exploratory Analysis



From Figure 1, we see there exists large class imbalances. We see Walking and Jogging take up the majority 38.86% and 30.92% respectively, however all other classes have much lower representations, with Sitting and Standing having approximately 5% each. Based solely on this, we may expect models to struggle with these lower balanced classes.

Explanatory Analysis

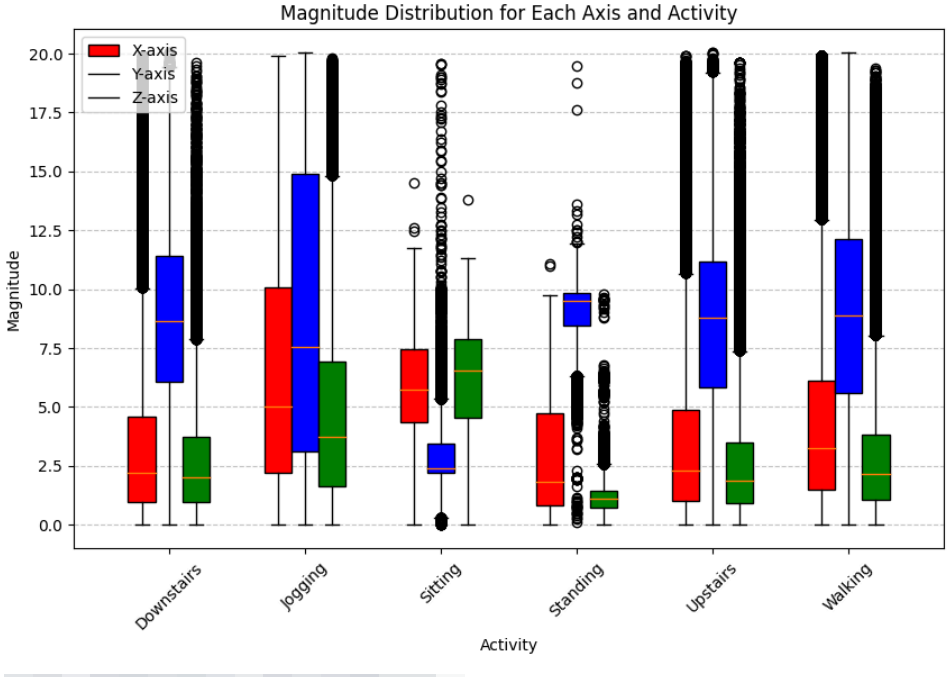


(a) Jogging (b) Walking

(c) Walking downstairs (d) Walking Upstairs

(e) Sitting (f) Standing

We then explored what the activities looked like. Figure 2 shows an example time series for each of the six activities. Immediately, we see drastic differences in the structures of the waves in Sitting and Standing. These are static activities; hence we expect to see very little variation in the values in the waves. Particularly, the z values on these differ greatly. On the other hand, the other activities are a lot harder to discern by eye.



When comparing the magnitude distributions of the activities other patterns emerge. We see that Downstairs Upstairs and walking all have very similar magnitudes, however we do see variation in the Jogging Sitting and Standing distributions. Jogging is unique in the fact that all three coordinate values have a relatively large IQR, whereas sitting is the only class that has the y coordinate as the lowest over average. Standing's unique identifier could be the lack of variation in the y and z values, having relatively small IQRs. All these unique characteristics could help the classification models differentiate between groups and improve accuracy of the prediction.

Using t-SNE, we attempt to visualise the clusters of data. For example, we see clear distinct clusters for Sitting, and there exist large clusters for Sitting and Jogging. However, we also see areas where lots of overlap exists. This is particularly evident for the Walking clusters. For example, in the centre of the plot, there is a cluster with Downstairs, Upstairs and Walking. We might expect models to struggle with distinguishing between these three especially.

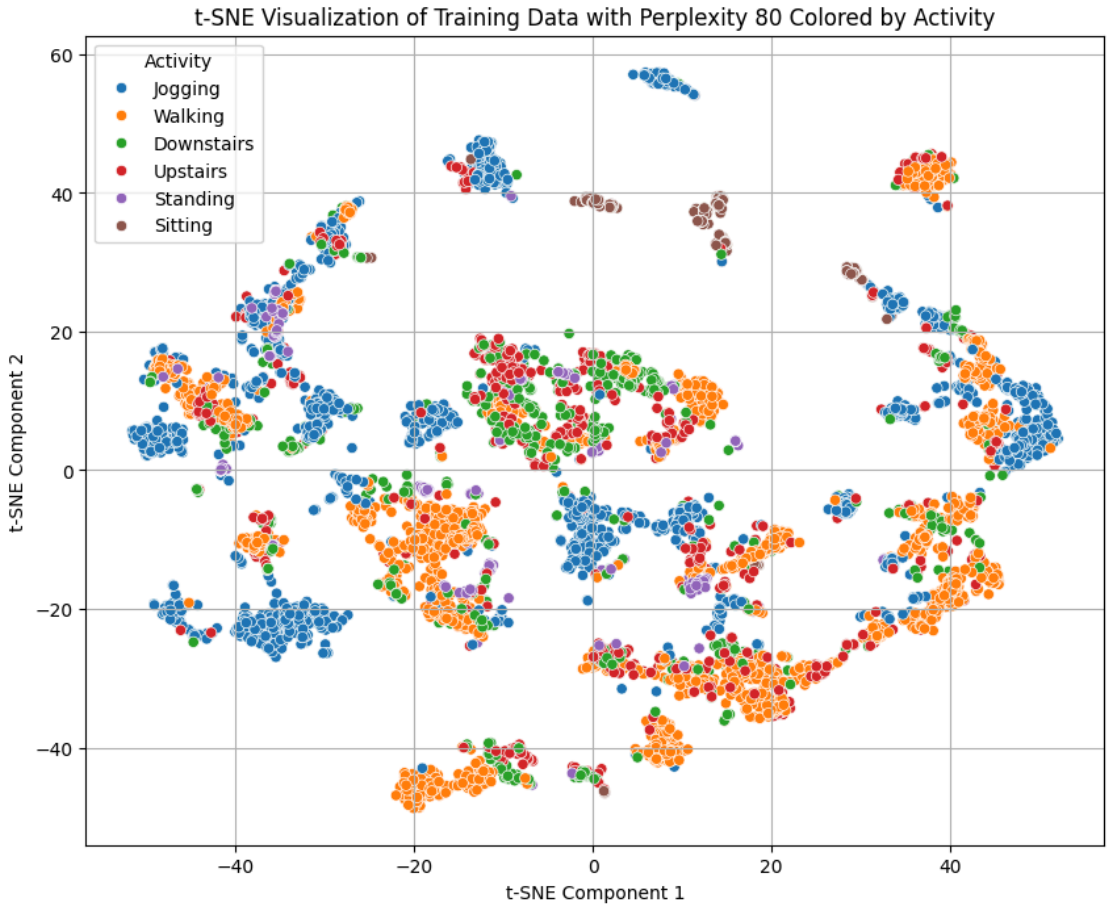


Figure 4: t-SNE visualisation of activities.

Feature Extraction

As previously explained, the original metadata dataset containing the simple summary statistics did not give enough detail about the accelerometer signal data to train powerful classification models. To

fix this we calculated further statistical values to add to the features in the metadata, for example, we added columns for trimmed mean, skewness, kurtosis and IQR (Interquartile Range).

Another feature we looked at was the correlations between the variables. We compute the Pearson's correlation coefficients,

$$\rho = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

#

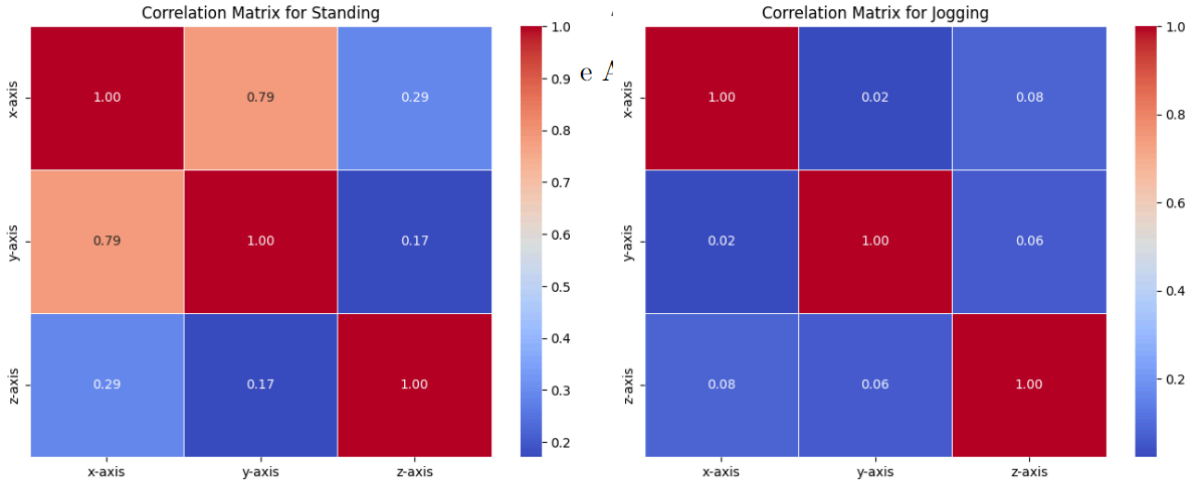


Figure 5: Correlation Matrices for Standing and Jogging

When plotting the correlation matrices, we find that there are strong correlations shown in the standing time series whereas very low correlation on average in the jogging time series. This dramatic difference in classes suggested that this would be a good feature to add as it would show significant variation between classes.

Other features extracted were more nuanced statistics each as average magnitude and difference in positive and negative, we were expecting these features to give less information to the mode however even small amounts of information can improve the model. In addition, we characterised the shape of the time series wave functions through the extraction of the axis peak values and their frequency.

After acquiring the extra features, we transform the data to increase the number of features. We first calculated the Fast Fourier Transformation (FFT). This is a transformation which computes the discrete Fourier transformation of signal data. It is given by the following formula.

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi \frac{kn}{N}}$$

With these transformations, we then computed the same statistics as before. This gave us our new complete metadata dataset.

Feature Selection

To select the important features for use, we decided to implement a decision tree-based method. With the in-built feature importance function within Python's Sklearns package, after implementing a random forest on our dataset, we can find which features contributed most to the trees. This will allow us to remove variables with low importance. We then iteratively tested the model removing one variable at a time, recording the accuracy each time, to find how many variables we should take in our final model.

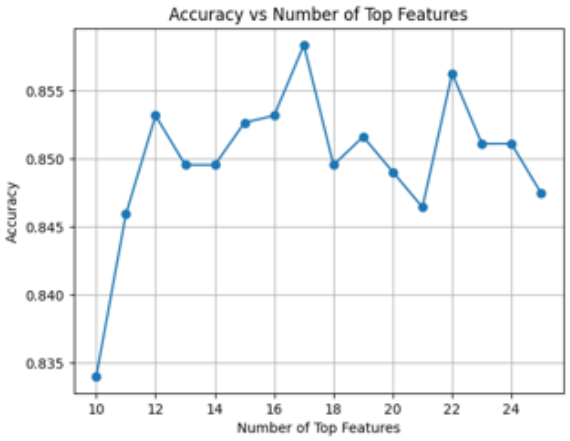


Figure 6: Accuracy vs Number of Important Features

As you can see the model with the highest accuracy had 17 variables. To examine which variables, we had in our final model we plotted the important features. We will use these as our extracted features for all future models.

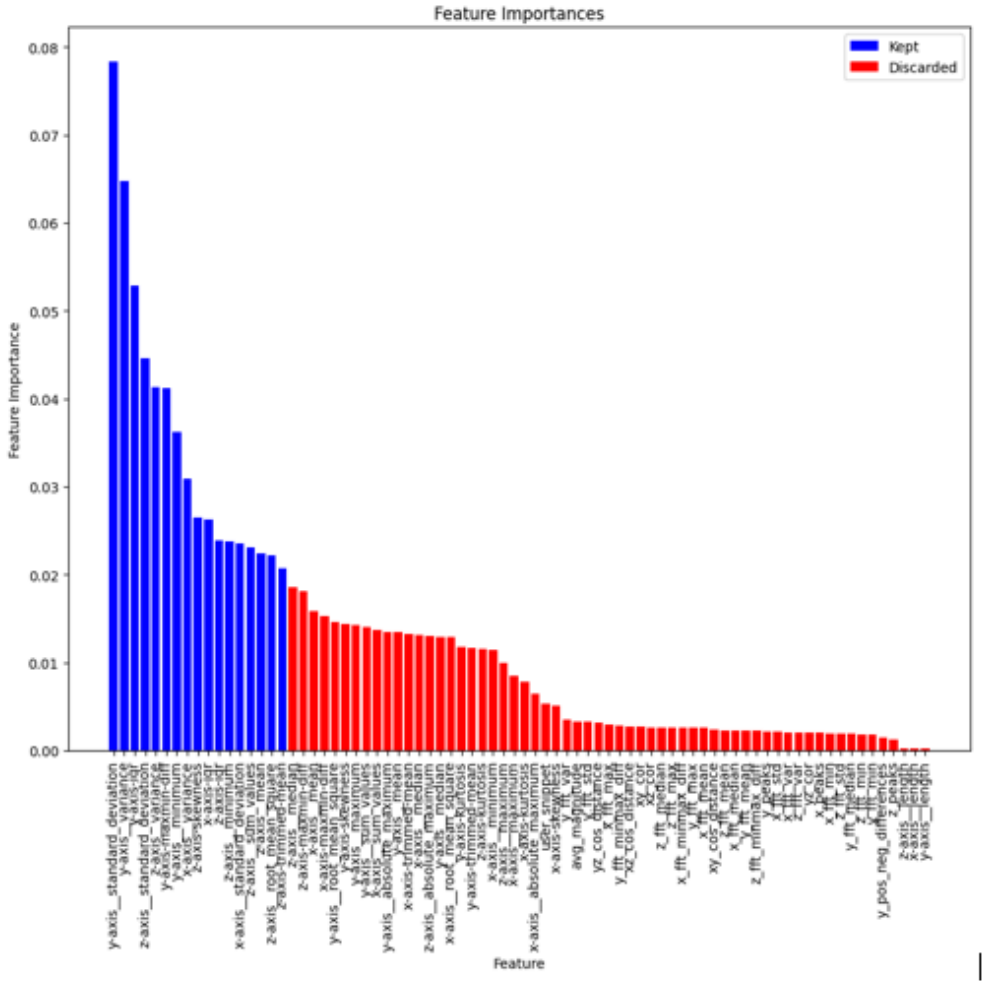


Figure 7: Feature Importance Graph

Taken Features:

- y-axis__standard_deviation
- y-axis__variance
- y-axis-iqr
- z-axis__standard_deviation
- y-axis-maxmin-diff
- z-axis__variance
- y-axis__minimum
- x-axis-iqr
- x-axis__variance
- z-axis-skewness
- z-axis__mean
- z-axis__minimum
- z-axis__root_mean_square
- z-axis-trimmed-mean

- z-axis__sum_values
- z-axis-iqr
- x-axis__standard_deviation

As we can see, within the 17 variables extracted, there exist 6 features which we extracted. These are the interquartile ranges, and the skewness in the z direction and y-axis-maxmin difference. This shows that the variables we extracted were useful in defining the decision boundaries and gained insight from the data. Another point of interest is that out of the 17 most important variables, only 3 were in the x direction, this is suggesting that the x coordinate time series were the most similar with each other and helped the least when classifying the activities. On the other hand, the y axis is the most important.

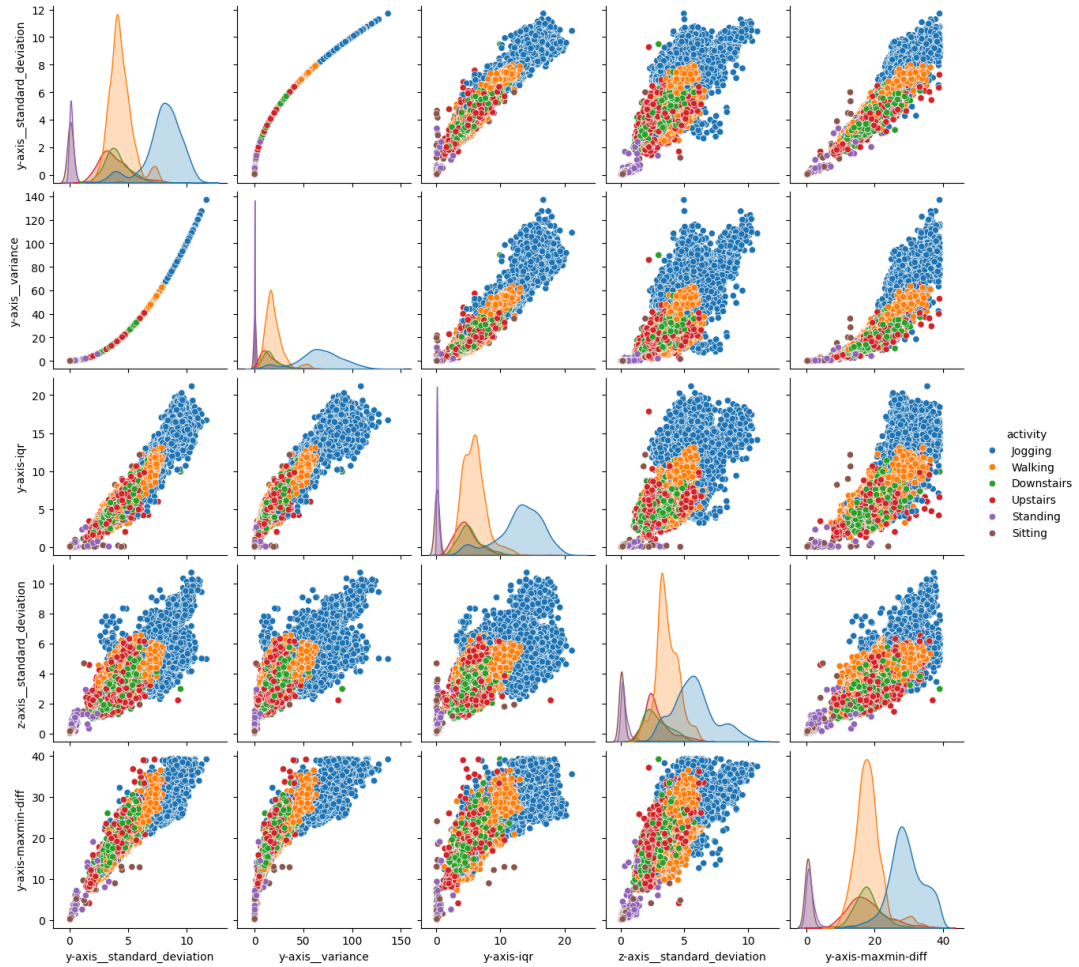


Figure 8: Pairs plot of 5 Most Important Features

To analyse the relationship between the most important features more closely a pairs plot (figure 8) of the five most important features was plotted with the probability distributions down the middle diagonal. The first point of interest is the plot of variance and standard deviation, obviously as they are measuring similar characteristics of the data, so they form a well-formed positive correlation. This plot has clear group differences thus will help the model in prediction, we see that jogging had higher variation whereas activities like upstairs sitting and standing had lower variation. Other notable features of the plot are jogging has higher values of iqr z standard deviation and max min diff, compared to the lower levels in standing. Walking had the tallest distributions showing the lowest variance and having values around the mean in all features. We see that jogging is

consistently high in all values, and Standing consistently low, with Downstairs, Upstairs and Walking littered around the midsections of each plot. Generally every scatter plot has a strong positive correlation, with some difference in variance. We see the similar shape in all scatter plots with jogging showing huge values and sitting with the lowest, these peaks show clear differences whereas the middle groups show a lot more overlap.

Random Forest Implementation

To implement the model, we used Python's Sklearn package, this package has inbuilt functions for all machine learning models and metrics plots, allowing the construction of the models to be as seamless and easy as possible. The first step in creating any machine learning model is to perform a grid search on the hyperparameters to systematically find the best ones for our dataset. This method will run a model with all combinations of specified hyperparameters, record the accuracy of all the models and select the best combination. This is important as it stops the models being under or over fitted to the data, if the model is too simple, the points will be miss represented, if the model is too complex it will be too strictly fitted to the training set and will not correctly adapt to the test set. This is the inherent trade off when constructing a machine learning model, known as the bias-variance trade off. The hyperparameters we checked through the grid search were:

```
param_grid = {
    'n_estimators': [20, 30, 40, 50, 100, 200, 300],
    'max_depth': [None, 10, 20,25],
    'min_samples_split': [5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
}
```

From this we found that the hyperparameters which gave us the most balanced model and achieved the greatest accuracy was:

```
max_depth=25,max_features='log2',min_samples_leaf=2,min_samples_split=5,n_estimators=300
```

```
Test Accuracy: 0.8304033092037229
```

For more proof the gridsearch has been implemented correctly, we can inspect plots of how the accuracy scores change when changing each hyperparameter individually.

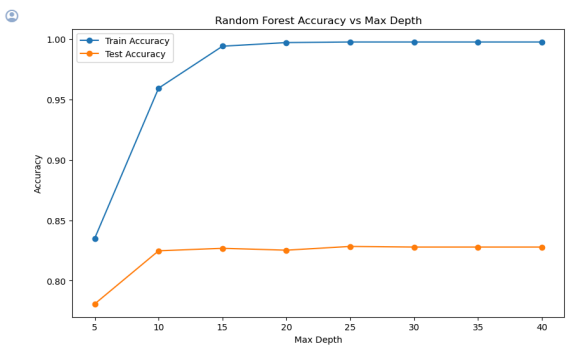


Figure 9: Accuracy vs Max Depth

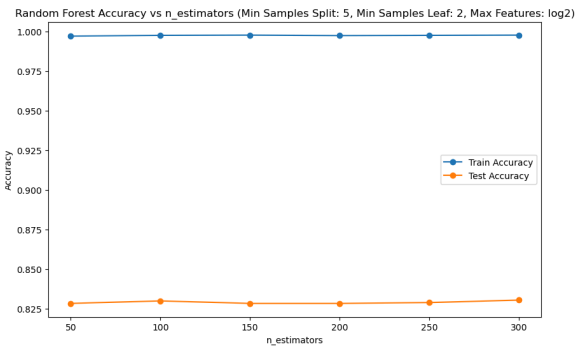


Figure 10: Accuracy vs n_estimators

The graph on the left shows the accuracy scores when periodically increasing the max depth of the random forest in increments of 5. In both the test and train scores there is a sharp increase in accuracy, then leading to a plateau at around 15, with minor differences in accuracy from then on. As we see no significant difference at higher values we can assume the max value found in the graph is the best max depth to choose, thus we use 25 as the parameter value. The graph on the right shows the variation in scores for the n_estimators parameter, throughout the whole grid search very little change in accuracy was recorded suggesting that this hyperparameter does not significantly change the decision boundary of the model, as a result we again chose the maximum of this graph at 300.

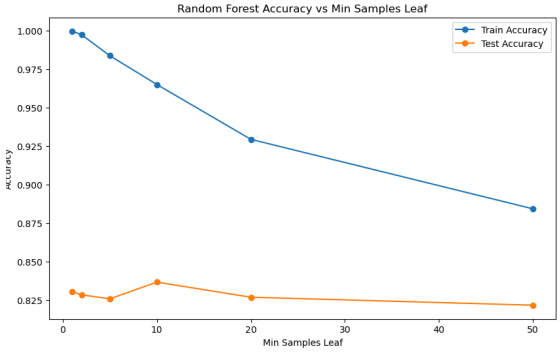


Figure 11: Accuracy vs Min Sample Leaf

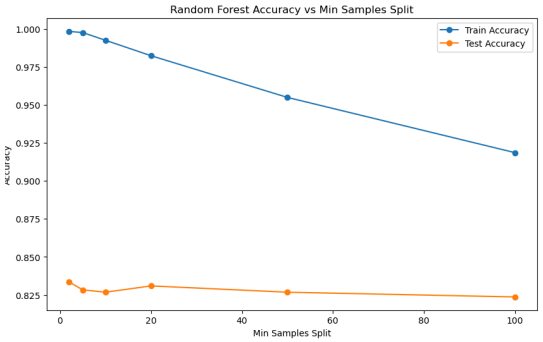


Figure 12: Accuracy vs Min Samples Split

Next we compared the accuracy scores of the min_samples_leaf and min_samples_split hyperparameters, these parameters change the way the decision trees are constructed and thus adhere to similar patterns, as both parameter values increase the accuracy in the training set falls dramatically, suggesting that we should chose a lower value. The test set shows more stability than the training, however the maximum value is still on the lower end. As a result, we know we can select 2 and 5 respectively.

After rerunning the model with our important features discussed earlier the accuracy of the model increased significantly, moving from 0.83 to 0.856. This improvement was due to the less important features teaching the model some bias or false patterns in the data that weren't there, thus streamlining the model and allowing prediction accuracy to increase. We used a k-folds cross validation on the model, with k = 10, which gave a slight decrease in accuracy. This may suggest that the model is overfitting.

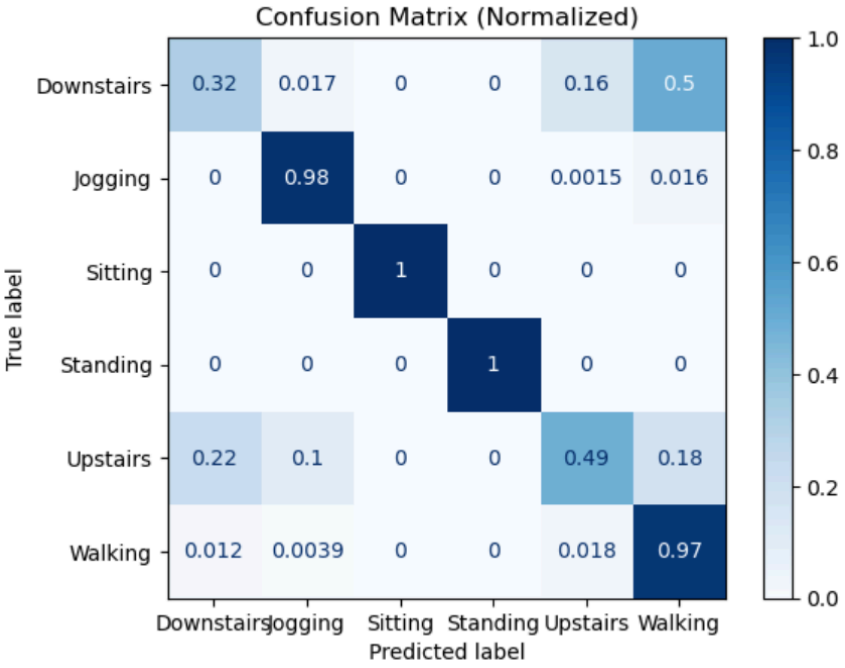


Figure 13: Confusion Matrix for Random Forest

Activity	Precision	Recall	F1-Score	Support
Downstairs	0.47	0.32	0.38	174
Jogging	0.96	0.98	0.97	689
Sitting	1.00	1.00	1.00	22
Standing	1.00	1.00	1.00	43
Upstairs	0.73	0.49	0.59	238
Walking	0.84	0.97	0.90	768
Accuracy			0.86	1934
Macro Avg	0.83	0.79	0.81	1934
Weighted Avg	0.84	0.86	0.84	1934

Figure 14: Classification Report for Random Forest

When looking at the classification report for this improved model we see that overall, we had an accuracy as 0.85, with a macro average of 0.80 and a weight average of 0.84. The macro average records the metric independently for each class and then takes the average of those metrics. It is suitable when you want to know how well the model performs across all classes without considering class imbalance. On the other hand, weighted average uses the same majority vote system but is calculated with weights corresponding to the number of instances in each class, it gives more weight to classes with more instances. Overall, however, using the standard accuracy gives a good overview of the prediction rate. Another point of interest is the Precision score, recall score and f1 score, the precision score measures the amount of correct positive predictions and is calculated through the ratio of true positive and false positive predictions, the recall score is slightly different, it measures how well the model could correctly identify the positive outcomes, being calculated through the ratio of true positives to false negatives. The f1 score is a combination of the precision and recall into one value.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad F1 \text{ score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Due to our imbalanced classes, the weighted average is the more useful metric to look at, and when comparing precision and recall through this we find that the scores are very similar, but the recall is slightly higher than the precision. This is suggesting that the model was slightly better at predicting true positives, however the difference is minimal. When comparing each activity's F1 score individually, we find that generally the model succeeded at predicting walking, standing, sitting and jogging, with 0.90, 1.00, 1.00 and 0.97 respectively. In contrast, the model struggled when predicting Downstairs and Upstairs with 0.38 and 0.59, as a result the overall accuracy of the model will be reduced.

When looking closer at the confusion matrix we can see where the model failed to predict correctly. We find that Downstairs the first activity misdiagnosed was being mistaken for upstairs and walking, and the second problem variable upstairs was being misdiagnosed as jogging downstairs and walking. In fact, downstairs is being predicted more often as walking than as itself, showing major issues in the classification.

The next model which provided good results for the metadata was a Convolution Neural Network (CNN). This is a model which is commonly applied to Image data, however not exclusively. Within layers, it treats the dataset as an image, and attempts to find features from this. Hence, we created a model with the following parameters:

Layer (type)	Output Shape	Param #
Conv1D (conv1d)	(None, 12, 32)	224
MaxPooling1D (max_pooling1d)	(None, 6, 32)	0
Conv1D (conv1d)	(None, 4, 32)	3104
MaxPooling1D (max_pooling1d)	(None, 2, 32)	0
Flatten (flatten)	(None, 64)	0
Dense (dense)	(None, 128)	8320
Dropout (dropout)	(None, 128)	0
Output Layers		
Dense (dense)	(None, 6)	774
Total params: 12422 (48.52 KB)		
Trainable params: 12422 (48.52 KB)		
Non-trainable params: 0 (0.00 Byte)		

Hyperparameters	
Epochs	40
Batch Size	32
Learning Rate	0.001
Drop Rate	0.5

This was quite a simple neural network, with only 12422 total parameters. It took two convolution and two pooling layers, which were pooled with max pool function, flatten and dense, with a drop out function of 0.5 to help prevent overfitting. We used a small batch size of 32, which provided the highest results from a grid search. Along with this, we used a validation split of 0.2 from the training data. We think this was sufficient balance between maintaining the training dataset's size and ensuring the effectiveness of the validation set. This is also true for all further models discussed.

As we see, we managed to slightly improve the accuracy of the test set up to 87.0. However, one thing to note is the way we did this. To achieve this increase in accuracy, we had to run for 40 epochs, however we see this caused a large increase in the validation loss. Hence despite performing better in our specific test set, we could assume it may not perform well on other unseen data due to overfitting. Hence, if we were to use only statistics data for classification, we would recommend our random forest model instead. We saw increases in the F score of our two weakest Activities, Downstairs and Walking, however this came at a cost to the Standing Score. We see it is significantly worse at recall.

Activity	Precision	Recall	F1-Score	Support
Downstairs	0.55	0.61	0.58	174
Jogging	0.95	0.98	0.97	689
Sitting	0.96	1.00	0.98	22
Standing	1.00	0.84	0.91	43
Upstairs	0.90	0.50	0.65	238
Walking	0.86	0.94	0.90	768
Accuracy			0.87	1934
Macro Avg	0.87	0.81	0.83	1934
Weighted Avg	0.87	0.87	0.86	1934

Figure 16: Classification Report for Feature CNN

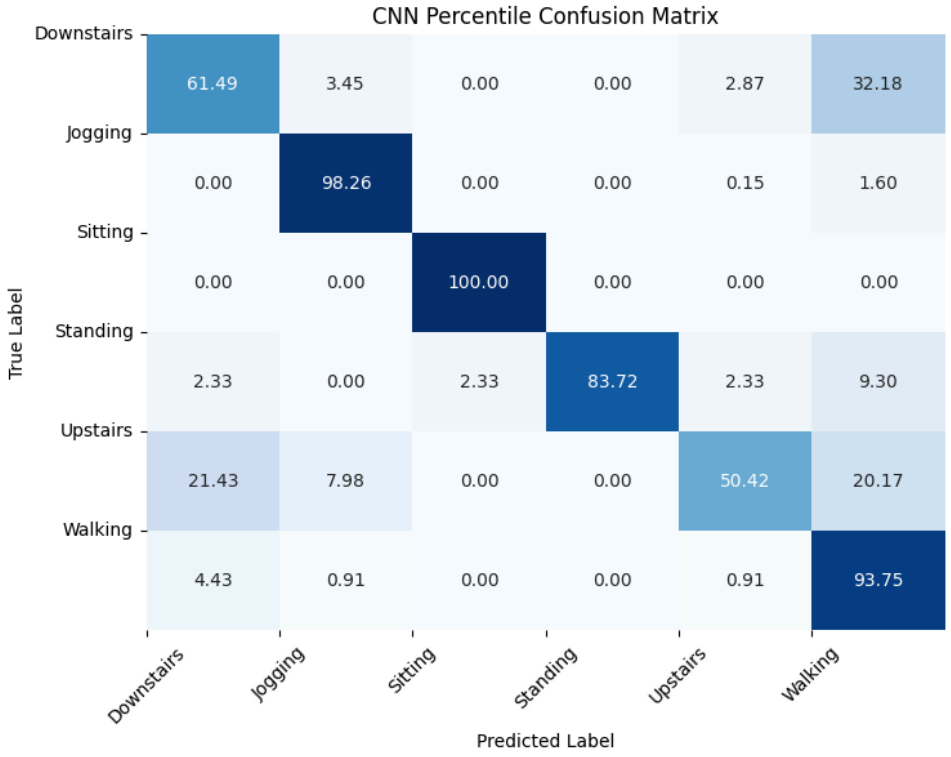


Figure 17: Percentile Confusion Matrix for Feature CNN

During experimentation, we tried to implement various models, including the Support Vector Machines, k-Nearest-Neighbours and Logistic regression. However, it was evident that the results being achieved were not ideal. Despite attempts to do hyperparameter tuning to optimise the models, the performance remained inferior to that of the Random Forest, with over five percent worse classification accuracy. Due to this, we made the decision to move away from these models, to focus on trying to improve other ones. It is important to acknowledge that the scores we received may not be the best achievable, however due to time constraints we thought it would be more beneficial to focus on the better scoring models. Scores achieved with all models attempted with the feature data is shown in the Figure 18.

Model	Test Accuracy (%)
Logistic Regression	77.5
K-Nearest-Neighbour	80.0
Random Forest	**85.6**
XGboost	84.9
Support Vector Machine	77.7
FNN	84.0
CNN	**87.0**

We found this was the limit of accuracy we could achieve using only the extracted features we had, hence instead we decided to turn our attention to implementing models with the time series data itself.

Time Series Models:

To seek to improve our classification accuracy, we then turned to look at the time series data. We first needed to ensure the data was in a form we could take. The first thing we looked at was the length of the snippets. The majority of the activities had 100 measures of x,y and z coordinates for each instance, spanning timestamps between 0-4950. However, when investigating, we noticed for each data set, including the training, the test and the Kaggle set, there existed cases where this was not adhered, with various measurements lower. However, the maximum number was 100. Hence, to ensure compatibility with various deep learning models which could be given to various deep learning models, we need to ensure the windows are a consistent size. To do this, we decided to pad these cases with zeros.

Following this, we similarly expanded this data set by computing the Fast Fourier transformation again. Finally, we then normalised the data, to ensure that they were scaled. We then dropped the time stamps from each instance. From this, we obtained our new transformed signal data set. To have it in a form the models can take, we convert these to an array. Hence, we had a training set with shape (6310, 100, 6). Finally, we ensured the signal data and the activities for each user were in the correct order.

The first model we used was a similar CNN to before, which immediately achieved better results, with an accuracy of 0.88. However, we chose to instead turn attention towards models more known for their ability with time series data.

One model we used was a Long Short-Term Memory (LSTM) neural network. This is an improved recurrent neural network; known for its use in sequential time series data. LSTMs have specialised memory cells, which allow them to capture long range dependencies within the data. Our specific LSTM had four LSTM layers, with originally 160 inputs, each decreasing down to a lowest 64 inputs. Following this, they were pooled, applied to a drop out of 0.5 to prevent overfitting. This gave us a model with 382,854 total parameters, significantly more than the CNN prior. We used a small grid search, due to the computational time, to find the Batch size, Learning Rate and Drop Rate we wished to use, and decided on Epochs based on the loss graph.

Layer (type)	Output Shape	Param #
LSTM	(None, 100, 160)	106,880
LSTM	(None, 100, 128)	147,968
LSTM	(None, 100, 96)	86,400
LSTM	(None, 100, 64)	41,216
GlobalMaxPooling1D	(None, 64)	0
Dropout	(None, 64)	0
Dense	(None, 6)	390
Total params:		**382,854 (1.46 MB)**
Trainable params:		**382,854 (1.46 MB)**
Non-trainable params:		**0 (0.00 Byte)**
Hyperparameters		
Epochs	25	
Batch Size	32	
Learning Rate	0.001	
Drop Rate	0.5	

Figure 19: LSTM Time Series Model

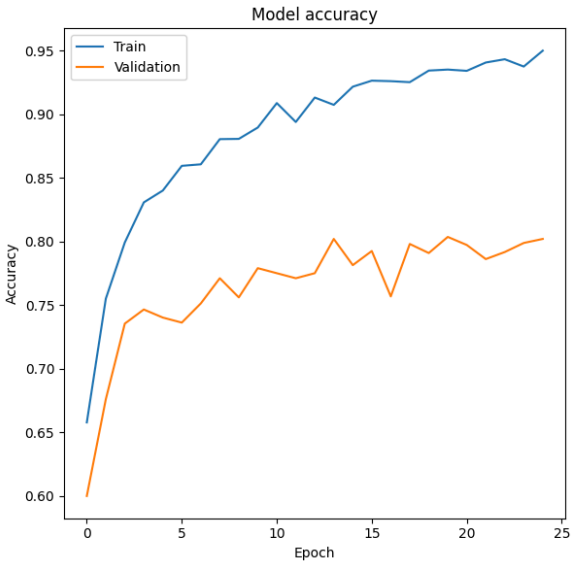


Figure 20: Model Accuracy

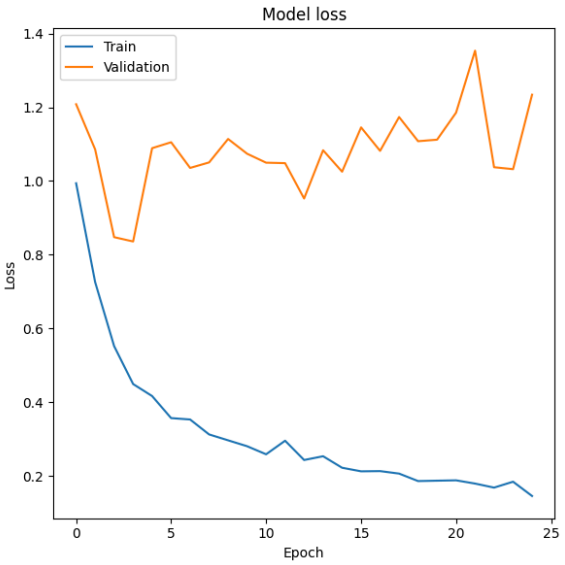


Figure 21: Model Loss

We see a huge increase in accuracy, with a weighted average of 0.91. The precision and recall of the Downstairs and Upstairs both increased significantly compared to previous models. Although we encountered some issues with the Recall of Standing. Looking at the confusion matrix, the model still struggles in the same areas as before, and in fact has gained issues in identifying Standing, which was in the random forest perfect. This may suggest slight overfitting. One thing we see is the increase in weighted accuracies. The random forest and CNN model both saw a decrease from their accuracy to their weighted accuracies, due to their struggles with certain classes. Whilst the issues in classification still exist, it performs a lot better.

Activity	Precision	Recall	F1-Score	Support
Downstairs	0.75	0.68	0.72	174
Jogging	0.96	0.98	0.97	689
Sitting	0.85	1.00	0.92	22
Standing	1.00	0.88	0.94	43
Upstairs	0.81	0.71	0.76	238
Walking	0.92	0.96	0.94	768
Accuracy			0.91	1934
Macro Avg	0.88	0.87	0.87	1934
Weighted Avg	0.91	0.91	0.91	1934

Figure 22: LSTM Classification Report

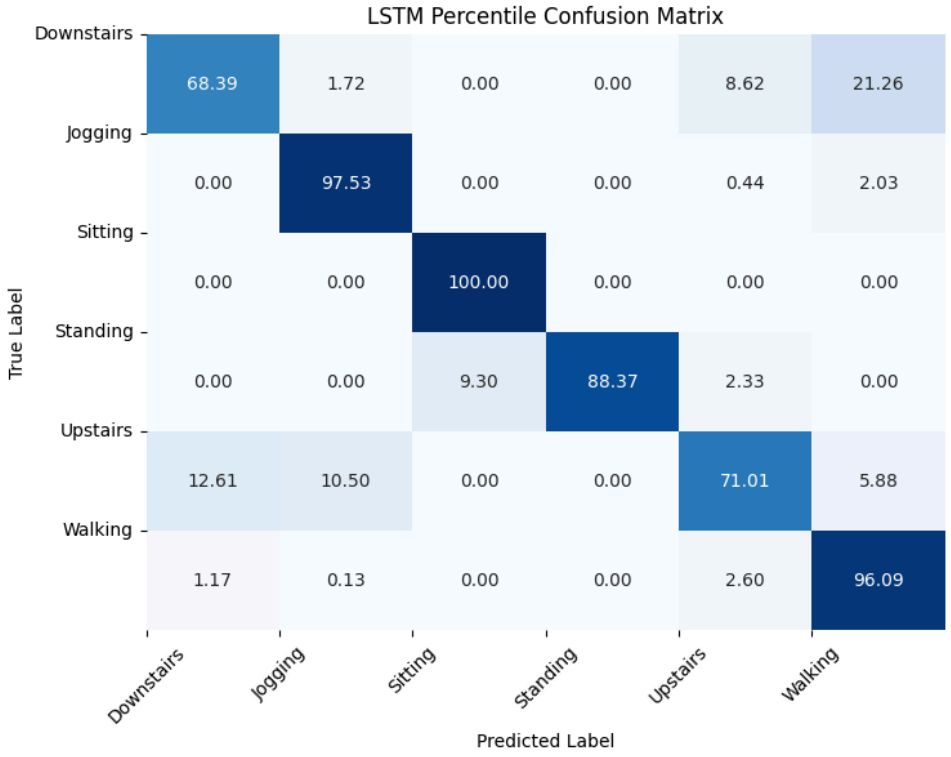


Figure 23: LSTM Confusion Matrix

Despite observing an increase in accuracy, our model encountered a significant rise in the loss function, rendering it less effective than anticipated due to overfitting. Another issue we faced with the LSTM was the time needed to run, due to the computational complexity. Hence, we chose to look at Gated Recurrent Unit (GRU) models. These are simplified LSTM, which only use on “gated” unit. Additionally, we chose to explore bidirectionality. This processes the sequence before forward and backwards, leveraging contextual information from both past and future timesteps. This potentially

could enhance its ability to capture patterns within the data. The mode we implemented is shown here:

Layer (type)	Output Shape	Param #
InputLayer	(None, 100, 6)	0
Bidirectional	(None, 100, 512)	405,504
BatchNormalization	(None, 100, 512)	2,048
Bidirectional	(None, 100, 256)	493,056
BatchNormalization	(None, 100, 256)	1,024
Bidirectional	(None, 100, 128)	123,648
BatchNormalization	(None, 100, 128)	512
Flatten	(None, 12,800)	0
Dense	(None, 128)	1638528
BatchNormalization	(None, 128)	512
Dropout	(None, 128)	0
Dense	(None, 6)	774
Total params		**2,665,606**
Trainable params		**2,663,558**
Non-trainable params		**2048**
Hyperparameters		
Epochs	10	
Batch Size	128	
Learning Rate	0.001	
Drop Rate	0.4	

Figure 24: Bi-Gru Model Summary

After taking our initial shaped input, it takes the bidirectional layer, with 512 neurons, 256 in each direction. We chose to implement batch normalisation layers this time, which help improve training speed, and apply regularisation to the data. This occurs for three layers, after which it is flattened, dense and normalised one more time. A final dropout rate of 0.4 occurs. Like the previous model, we hyperparameter tune to get our dropout, batch Size and learning rate. We found that 10 epochs were sufficient in training and preventing overfitting.

Activity	Precision	Recall	F1-Score	Support
Downstairs	0.76	0.66	0.70	174
Jogging	0.93	0.98	0.96	689
Sitting	0.92	1.00	0.96	22
Standing	1.00	0.93	0.96	43
Upstairs	0.90	0.66	0.76	238
Walking	0.92	0.98	0.95	768
Accuracy			0.91	1934
Macro Avg	0.90	0.87	0.88	1934
Weighted Avg	0.91	0.91	0.91	1934

Figure 25: Bi-GRU Classification Report

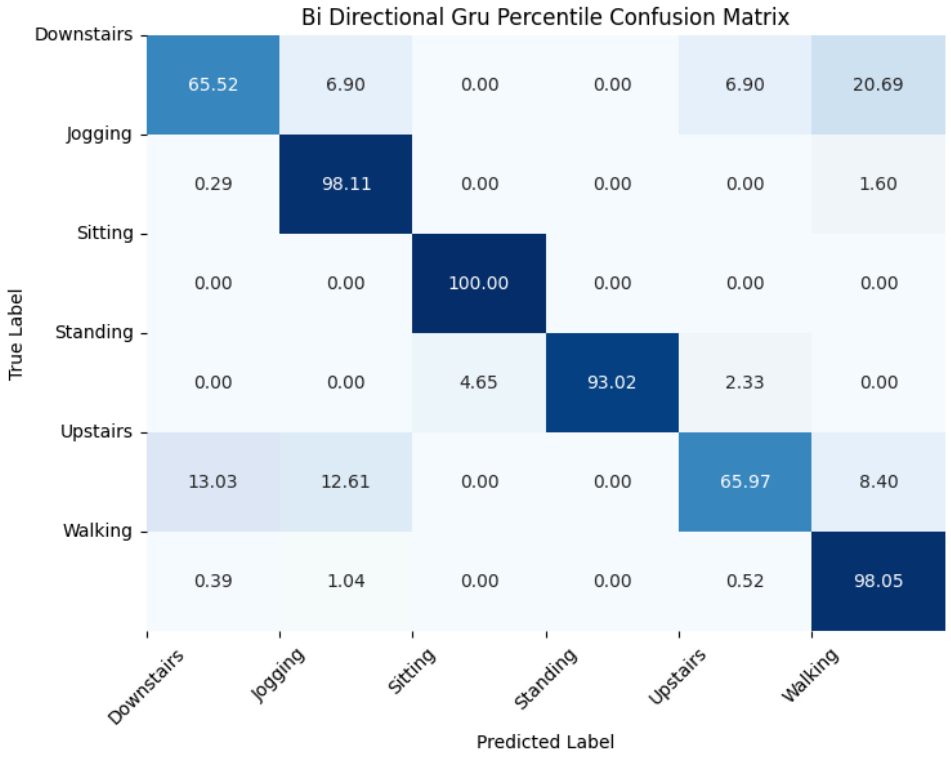


Figure 26: Bi-GRU Confusion Matrix

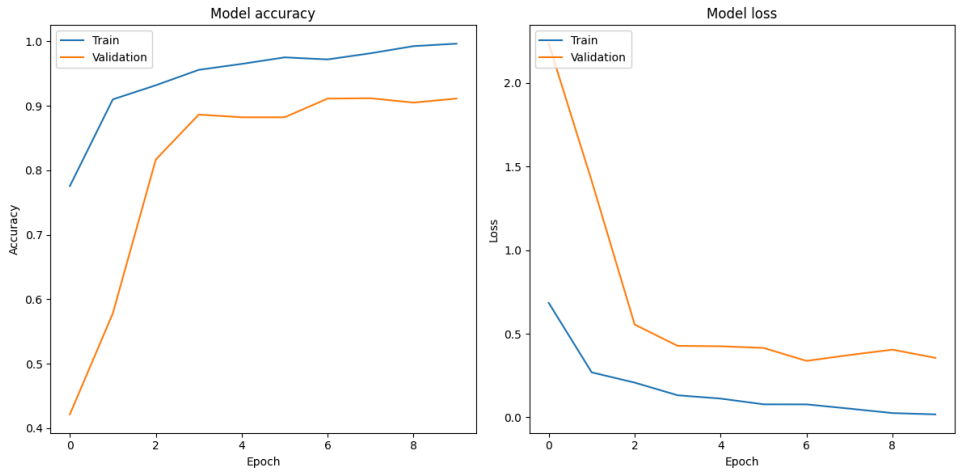


Figure 27: Model Accuracy vs Epoch

Figure 28: Model loss vs Epoch

We see that whilst it performed very similarly to the LSTM model, it has slightly lower F1 scores for Downstairs, and the same for Upstairs. The main benefit to this model is the much lower model loss, indicator it is less overfit and will perform better on more unseen data.

Combined Statistics and Time series.

With initial time series models, we saw significant improvements compared to using just feature extracted items however to seek additional improvements, we wish to see what would happen when combining both. First, we reshaped our metadata with the important features we found before into the same array format and altered the deep learning models to have separate layers for each.

Taking inspiration from previous models built, we were able to create combined models for CNN models, LTSM and GRU. The highest accuracy we achieved was with the conv hybrid model. This is shown in Figure X. Our model consists of two separate branches. The first was the convolution layer, which initially had 128 filters and a kernel of size 10. This was then pooled and convolved two further times with layers of decreasing size. The max pooling layer then extracted the key features.

Layer (type)	Output Shape	Param #
Time Series Branch		
Conv1D (conv1d_134)	(None, 91, 128)	7808
MaxPooling1D (max_pooling1d_69)	(None, 45, 128)	0
Conv1D (conv1d_135)	(None, 38, 64)	65600
MaxPooling1D (max_pooling1d_70)	(None, 19, 64)	0
Conv1D (conv1d_136)	(None, 14, 32)	12320
GlobalMaxPooling1D (global_max_pooling1d_42)	(None, 32)	0
Statistics Branch		
Dense (dense_114)	(None, 256)	4608
Dropout (dropout_77)	(None, 256)	0
Concatenate (concatenate_35)	(None, 288)	0
Output Layers		
Dense (dense_115)	(None, 128)	36992
Dropout (dropout_78)	(None, 128)	0
Dense (dense_116)	(None, 6)	774
Total params: 128102 (500.40 KB)		
Trainable params: 128102 (500.40 KB)		
Non-trainable params: 0 (0.00 Byte)		
Hyperparameters		
Epochs	35	
Batch Size	64	
Learning Rate	0.0005	
Drop Rate	0.4	

Figure 29: CNN Time Series / Features Hybrid Model Summary

After this, the statistics branch has a dense layer with 256 neurons. This was then concatenated with the time series branch.

Finally, this was passed through additional dense layers, with drop out to prevent overfitting. This was then classified. We found the best hyperparameters such as the drop out and learning rate using a grid search, and altered the number of epochs to ensure that the loss function of the validation set was not increasing.

Looking at the figures for train and validation accuracy, we see that whilst the accuracy of the train set continuously increases, we see a levelling of the validation set around approximately 0.85. Similarly, we see the loss function begin to separate after 25 epochs.

Activity	Precision	Recall	F1-Score	Support
Downstairs	0.82	0.65	0.73	174
Jogging	0.95	0.98	0.96	689
Sitting	0.92	1.00	0.96	22
Standing	1.00	0.95	0.98	43
Upstairs	0.84	0.77	0.81	238
Walking	0.93	0.98	0.95	768
Accuracy			0.92	1934
Macro Avg	0.91	0.89	0.90	1934
Weighted Avg	0.92	0.92	0.92	1934

Figure 30: Hybrid Model Classification Report

Figure X showcases the classification report for this model. When compared to our original random forest model, we see huge increases in the precision and the recall of Downstairs and Upstairs, with the F score to 0.73 and 0.77 respectively. We see roughly consistent values for our Jogging, Sitting and Standing, as there was no room for much increase, and a small increase in walking. The average accuracy and weighted averages are all 0.92, showing a huge increase in our values compared to the random forest model.

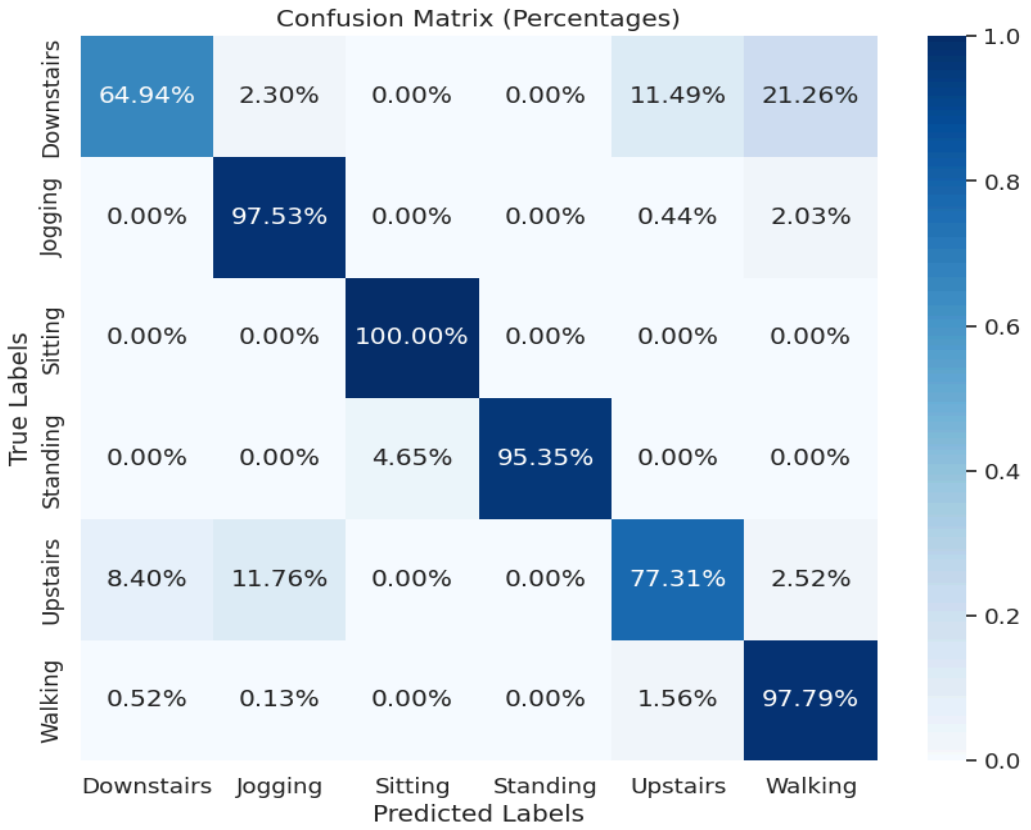


Figure 31: Hybrid Model Confusion Matrix

Investigating the confusion matrix, we can note that the model still struggles with the same problems. We see Downstairs is still predicted as both Upstairs and Walking, and it might be presumed that this may not entirely be able to be solved. However, we see huge improvements, for example only 21.26% of Downstairs is predicted as walking, compared to 48.28% in the random forest.

Conclusion:

To conclude, we have managed to create multiple models with varying degrees of accuracy and managed to improve classification from baseline models of 0.80 to 0.92 for our test data. No model was superior in every aspect, for example the LSTM was better at identifying “Downstairs”, but CNN with Time series and Statistics was better at “Upstairs”, thus the structure of the class imbalances in the test data could affect which model gave the best prediction accuracy as we only recorded marginal differences in the deep learning models. When evaluating all aspects, we found that our CNN Model was the best. This was also shown in our Kaggle score, being the model that provided the highest accuracy.

Model	Test Accuracy (%)
LSTM (Time Series)	90.8
CNN (Time Series/ Statistics)	**92.3**
Bi-GRU (Time Series)	91.2
Random Forest (Statistics)	86.0

Moreover, it is evident that the models which utilise the time series as the input had notably higher accuracies compared to the feature extracted models. This is as there is more nuanced information for the model to learn from. Whilst the feature extracted models aim to capture the important information, this is inherently weaker compared to the time series models.

We should note that the CNN model computed a lot quicker than both the LSTM and GRU models, hence more experimentation was performed with and had the opportunity to perform grid searches for parameters. This was not entirely feasible with the other models. We understand that perhaps given more time and experimentation, the combined other models may have increased accuracy, particularly now we are reaching the limit of only very small increases. Hence given more time, we would like to experiment with these models more.